Bolt Beranek and Newman Inc.

(12)

Report No. 5239

Research in Speech Understanding

Annual Report

December 1982

Prepared for:
Advanced Research Projects Agency

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE — READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER: BBN Report No. 5239
2. GOVT ACCESSION NO.: A123 218
3. RECIPIENT'S CATALOG NUMBER:

4. TITLE (and Subtitle): Research in Speech Understanding Annual Report
5. TYPE OF REPORT & PERIOD COVERED: Annual Report 10/1/81 - 9/30/82
6. PERFORMING ORG. REPORT NUMBER: BBN Report No. 5239

7. AUTHOR(s): Richard Schwartz, John Makhoul, Salim Roucos, A.W.F. Huggins
8. CONTRACT OR GRANT NUMBER(s): N00014-81-C-0738

9. PERFORMING ORGANIZATION NAME AND ADDRESS: Bolt Beranek and Newman, 10 Moulton Street, Cambridge, MA 02238
10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS:

11. CONTROLLING OFFICE NAME AND ADDRESS: Office of Naval Research, Department of the Navy, Arlington, VA 22217
12. REPORT DATE: December 1982
13. NUMBER OF PAGES: 55

14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office):
15. SECURITY CLASS. (of this report): Unclassified
15a. DECLASSIFICATION/DOWNGRADING SCHEDULE:

16. DISTRIBUTION STATEMENT (of this Report)

Distribution of this document is unlimited. It may be released to the Clearinghouse, Department of Commerce, for sale to the general public.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Speech Understanding, speech recognition, phonetic recognition, acoustic-phonetics, acoustic-phonetic features, context-dependent phonetic recognition.

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

This annual report describes the work performed during the first year of a two-year effort to develop improved techniques for acoustic-phonetic recognition, with eventual application to automatic speech understanding.

The work during this year was devoted to three major activities: speech database generation, computer tool building, and development of acoustic-phonetic rules. A large database of speech was designed and

DD FORM 1473, 1 JAN 73 — EDITION OF 1 NOV 65 IS OBSOLETE

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

then recorded by four male speakers. Many utility programs that had been developed for the PDP-10 TOPS-20 system were modified to run on the VAX 11/780 under VMS. We initiated the development of a set of context-dependent acoustic-phonetic features to distinguish among phonemes with greater accuracy.

Report No. 5239

RESEARCH IN SPEECH UNDERSTANDING

Annual Report
1 October 1981 to 30 September 1982

Principal Investigator:
John Makhoul
(617) 497-3332

Prepared for:

Defense Advanced Research Projects Agency
1400 Wilson Boulevard
Arlington, VA 22209

ARPA Order No. 4311 Contract No. N00014-81-C-0738

Effective Date of Contract: 1 October 1981

Contract Expiration Date: 30 September 1983

This research was supported by the Advanced Research Projects Agency of the Department of Defense and was monitored by ONR under Contract No. N00014-81-C-0738. The views and conclusions contained in this document are those of the authors and should not be interpreted as necessarily representing the official policies, either expressed or implied, of the Defense Advanced Research Projects Agency or the U.S. Government.

TABLE OF CONTENTS

1. OVERVIEW

This annual report describes the work performed during the first year of a two-year effort. The overall goal of the project is to develop improved techniques for acoustic-phonetic recognition, with eventual application to automatic speech understanding.

Earlier work on this subject has yielded phonetic recognition rates of 60-70%. We estimate that a minimum recognition rate of 80% is needed for a high-performance speech understanding system. We believe that the information needed to achieve the higher recognition accuracy is available in the speech signal. This research is directed towards discovering as many context-sensitive acoustic-phonetic rules as possible to improve the recognition performance.

This first year of research was devoted to three major activities: Speech database generation, computer tool building, and development of acoustic-phonetic rules. Specifically, the following tasks were performed:

1. Designed, recorded, digitized and processed a database of speech.
2. Converted and extended several large utility programs that had run on the PDP10 to run on the VAX 11/780.
3. Implemented spectrogram display programs for the Benson Varian printer and the BBN BitGraph display terminal.
4. Examined a selected number of spectrograms and catalogued acoustic-phonetic rules.
5. Initiated the development of a set of context-dependent acoustic-phonetic features to distinguish among phonemes with greater accuracy.

In the following sections, we describe each of these tasks in greater detail.

To aid in our acoustic-phonetic work, a real-time spectrograph display was purchased from Spectraphonics. The real-time spectrograph has been useful in looking at spectrograms and discovering certain acoustic-phonetic properties.

2. DATABASE

To enable us to discover rules for phonetic recognition that are applicable to a wide range of phonetic environments, we designed a large database of sentences. The initial database consists of 110 different sentences. Phonetic transcriptions of the sentences were analyzed statistically to ensure that they represented aphonetic balance similar to that in fluent everyday speech. Appendix 1 contains a list of the 110 sentences. It also indicates the word count and phoneme counts used in the phonetic analysis. Each sentence was recorded by four male speakers. The digitized speech waveforms were then analyzed using our Primary Speech Analysis (PSA) program, which had been converted to run on the VAX computer. The output of the PSA program is a large set of acoustic parameters such as the energy in different frequency bands, zero-crossing count, formant frequencies, etc., for each 10 ms frame. These parameters will be used by the Acoustic-Phonetic Recognition (APR) program that will be developed under this project.

We have just begun to align the phonetic transcription of each of the sentences with the parameters. This will allow us to use the Acoustic-Phonetic Experiment Facility (APEF) to discover more rules and features by looking at all occurrences of any phonetic sequence within the database.

To augment our new database, we have also transferred the phonetically transcribed database that had been used in the ARPA Speech Understanding Research (SUR) Project to the VAX.

3. UTILITY PROGRAMS CONVERTED

Our move from the PDP-10 computer to the VAX-11/780 has necessitated the conversion of all of our utility programs that had been developed over the past several years. The move to the VAX computer was necessary as the VAX system provides both real-time processing capabilities (with the aid of the FPS-AP120B and the DEC LPA-11K systems) and interactive computing on a computer with a large virtual address space. Both of these capabilities are necessary for the efficient performance of our research.

Below we describe the programs that have been transferred (and rewritten when necessary).

3.1 Library

We have, over the years, developed an extensive subroutine library with routines that perform a wide variety of signal processing functions, string manipulation functions, user interaction functions, etc. As a first step to developing programs on the VAX computer, we transferred or rewrote all of the generally useful subroutines in our user library. Appendix 2 contains a list of the routines in the library, with a short description for each.

To make the library more useful, we used the on-line documentation facility provided by the VAX VMS operating system, together with a tree structured documentation file.

3.2 RTIME

This program, which had resided on our PDP-11/40 system and was converted to run on the VAX, is used for real-time signal acquisition and playback, waveform editing, and spectral display. Although the original PDP-11 program and the new VAX program are both in FORTRAN, the program had to be almost completely rewritten due to the differences in I/O, real-time systems, FPS software package, and operating system facilities. The program was also modified to enable recording of large databases with improved user-machine interaction and automatic editing of silences.

One important type of speech database is characterized by large numbers of individual sentences. Extracting these from a previously recorded audio tape is a time consuming and laborious task. Consequently, we implemented a semi-automatic sentence acquisition capability into the RTIME program. In the most commonly used mode of operation, the program is given a list of file names for the output. The program then prompts the user with the name of the file and waits for a carriage return. The speaker types the carriage return, and whenever ready, speaks the sentence. The program automatically detects the beginning and end of speech, plays back the edited waveform, and asks for verification. If verified, it writes out the file and goes on to the next sentence. Otherwise, it allows many other operations such as playing the file again, recording it again, or allowing the user to override the end points chosen by the program.

3.3 PSA

The Primary Speech Analysis (PSA) program performs a wide variety of functions. First, it accepts as input a digitized speech waveform, and analyzes it to produce a large set (50) of acoustic parameters. The parameters computed include several energy parameters derived from the LPC spectrum, the formant frequencies (also derived by algorithm from the narrow-bandwidth LPC poles), and other useful parameters such as the zero-crossing count and spectral center of mass. The speech parameters and manual transcriptions produced by this program are then used by the APEF and phonetic recognition programs (described below).

The program is capable of displaying the most commonly used parameters using the IMLAC display computer. It also displays a section of the speech waveform and the LPC power spectrum at a time indicated by a cursor. By allowing the simultaneous display of all the available information, the program allows a user to develop intuitions for distinguishing among phonemes. The program is also used for the interactive manual transcription of speech material.

Since PSA consists largely of I/O functions and interactions with the IMLAC display computer, the program had to be completely rewritten. The program had evolved over a period of 10 years, and this was a good opportunity to restructure the many functions in a well-organized manner. The new program was written in FORTRAN-77, which is a block structured language, making this redesign straightforward.

3.4 APEF

The Acoustic-Phonetic Experiment Facility (APEF) is a highly interactive system that allows a speech researcher to perform a wide variety of experiments on a large database of continuous speech [1]. This system, which has been used extensively during our previous phonetic research, is essential to the development of a large set of acoustic-phonetic features and rules for recognizing the different phonemes of English.

APEF was written in BCPL on the PDP-10, so it had to be translated to a language that exists on the VAX. Several languages were considered for this translation, including PASCAL, PRAXIS, a subset of ADA, FORTRAN-77, and C. C was chosen since it is, by far, the language that is most similar to BCPL. C has recently become available on VAX VMS systems, and is compatible with all other VMS languages. The translation of 12,000 lines of BCPL code into 15,000 lines of C code has been completed. At present, some of this very large program has been debugged and is operating correctly. However, there are still several essential modules that have not been debugged as yet. This debugging process should be completed during the coming year.

Since this program will be used so extensively in the research under this project, we present below a brief description of its capabilities. The primary use of the program is to examine many occurrences of some set of phonetic phenomena extracted from a large database of continuous speech, and then to

examine this data statistically, and develop algorithms for phonetic recognition without the need to incorporate these algorithms in a complete phonetic recognition program. The user first specifies a phonetic context of interest. For example, he might be interested in distinguishing among the unvoiced plosives when followed by vowels or glides. The phonetic context is specified in terms of a sequence of segment descriptions. These descriptions can be given in terms of phonemes, classes of phonemes, or boolean combinations of classes. The program also allows the user to specify the context in terms of the presence or absence of word boundaries, syllable boundaries, different vowel stress levels, or particular words. It also allows for the designation of some of the segments being optional.

Once the phonetic context has been specified, the user enters a procedure that will compute several acoustic-phonetic features. The procedure is given in terms of a list of high-level routines. The routines include a wide variety (52) of functions for examining acoustic parameters (e.g., average, maximum, minimum, standard deviation, etc.), arithmetic functions (e.g., sum, difference, product, quotient, abs, max, min, etc.), boolean functions (e.g., and, not, or), loops, branches, tests, display functions, and computation of new acoustic parameters from the raw power spectrum.

The user then instructs the program to search the database for occurrences of the specified context. For each occurrence

found, the procedure is run to compute the desired features. Then, the user is given a variety of ways of examining the data gathered. The program is capable of producing various statistical displays on the IMLAC display processor, giving statistical information for a specified feature, and performing pattern recognition experiments on the gathered data to attempt to distinguish among the different phonemes using the features gathered.

The end result is that an initial experiment can be performed in just a few minutes, and easily modified, making it possible to fully investigate a particular phonetic recognition problem in a few hours.

3.5 IMLAC Display Interface

The IMLAC display interfaces that had been developed on the PDP10 were reworked so that the VAX could communicate with the IMLAC in the same ways as the PDP10. This conversion included the special purpose IMLAC programs used by RTIME (SPECIZ), which supports the scrolled waveform display and two spectrum displays. The PSA program also uses a special purpose IMLAC display program (PSAI), which allows the simultaneous display of 14 speech parameters, a power spectrum, and a section of a speech waveform. It also includes several 2-way interactions between the host computer and the IMLAC computer, controlled by various knobs and switches connected to the IMLAC. Finally, we brought up the

general purpose display package, IMSYS, that is used by APEF and other programs.

4. DISPLAY PROGRAMS

We have developed several display programs for the VAX computer. These display programs fall into two categories:

1. Spectrogram Displays
2. Speech Parameter Displays

We have developed a spectrogram display program that produces a hardcopy display of a spectrogram on the Benson Varian printer, which is connected to the VAX computer. This program uses the FPS for the signal processing and image processing necessary to produce a pleasing visual display. We have also developed a version of the program that produces a spectrogram on the BBN BitGraph terminal. These interactive programs are intended to complement the capabilities of the Spectraphonics Real-Time Spectrograph. In particular, the new programs also allow the display of several acoustic parameters adjacent to the spectrogram, on the same time scale, using the PARPLOT program discussed below.

The PARPLOT program was developed to make hardcopy displays of the speech parameters (computed by PSA) on the Benson Varian printer. These parameter displays are scaled such that they are compatible with the spectrogram displays. The program allows the user to specify, in an interactive way, which parameters are to be plotted, and with what scales. These hardcopy plots will be used to help design the acoustic-phonetic recognition algorithms.

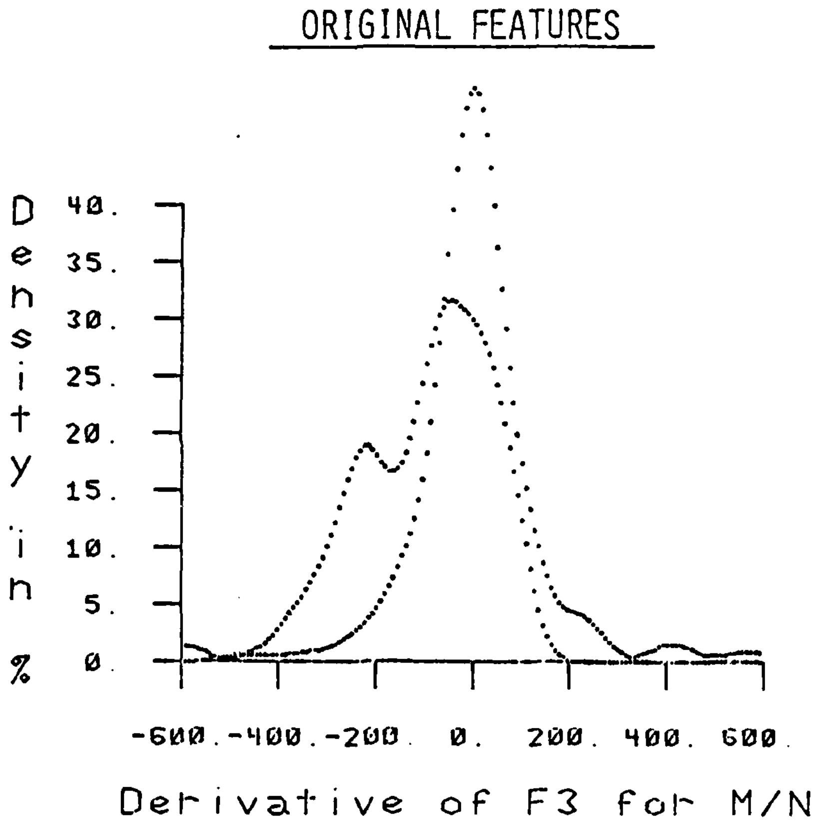


Fig. 1. RAW FEATURE HAS LARGE OVERLAP

5. ACOUSTIC-PHONETIC RULES/FEATURES

One of the primary research tasks performed this year was to develop context-dependent acoustic-phonetic rules for distinguishing among phonemes with greater accuracy than previously possible. For each class of phonemes, we began by examining spectrograms of those phonemes in different contexts. The Spectraphonics was particularly useful for this process, since it allowed us to generate spectrograms for a phoneme in many contexts with a minimum of effort. An important source of ideas for useful features was obtained from our interactions with the research group at MIT. When an idea was mentioned, we were able to verify it quickly, using the spectraphonics. Most often, the rules developed from looking at spectrograms are qualitative in nature, and are also based on a very small number of samples. Therefore, we then used APEF to develop specific quantitative rules and tested them on a large database.

While the VAX version of APEF is being translated and debugged, we have been using the PDP-10 version of the program with our old database of labeled speech to continue to find rules and features that are useful for acoustic-phonetic recognition. The rules that we are developing are of the type that would be appropriate for addition to the Acoustic-Phonetic Recognition (APR) program that is also being written. Most of the effort with APEF, however, has been directed towards finding features that are most useful for distinguishing among phonemes that are

within certain classes. We have found that it is possible to distinguish among particular phonemes within a class (for example among the three nasal phonemes, or the three unvoiced plosive phonemes) with 80%-90% accuracy. This accuracy is achieved by searching for features that are fairly independent of the phonetic context (adjacent phonemes), and also by considering separate phonetic contexts where appropriate.

Typically, the features are made independent of the phonetic context by normalizing them by some measurement made during the adjacent phoneme. For example, one strong cue to the identity of a nasal is the direction of the formant transitions in the preceding vowel. The second and third formants tend to drop preceding an [M], rise preceding an [N], and come together preceding and [NX]. However, the actual amount and direction of the transition depends on the previous vowel. Figure 1 shows density distributions of the derivative of F3 in the two frames preceding an [M] or an [N]. (The figures shown were produced by the APEF program and transferred directly from the IMLAC screen to the Benson-Varian printer plotter.) As can be seen, the overlap in the distributions for the two phonemes is quite large, making this (commonly quoted) feature quite useless. However, by normalizing the amount of formant change by the measured formant frequencies in the middle of the vowel, the transitions are made much more useful in distinguishing among the nasals, without having to know exactly what the preceding vowel was. Figure 2 shows density distributions of the adjusted derivative values.

CONTEXT-DEPENDENT FEATURES

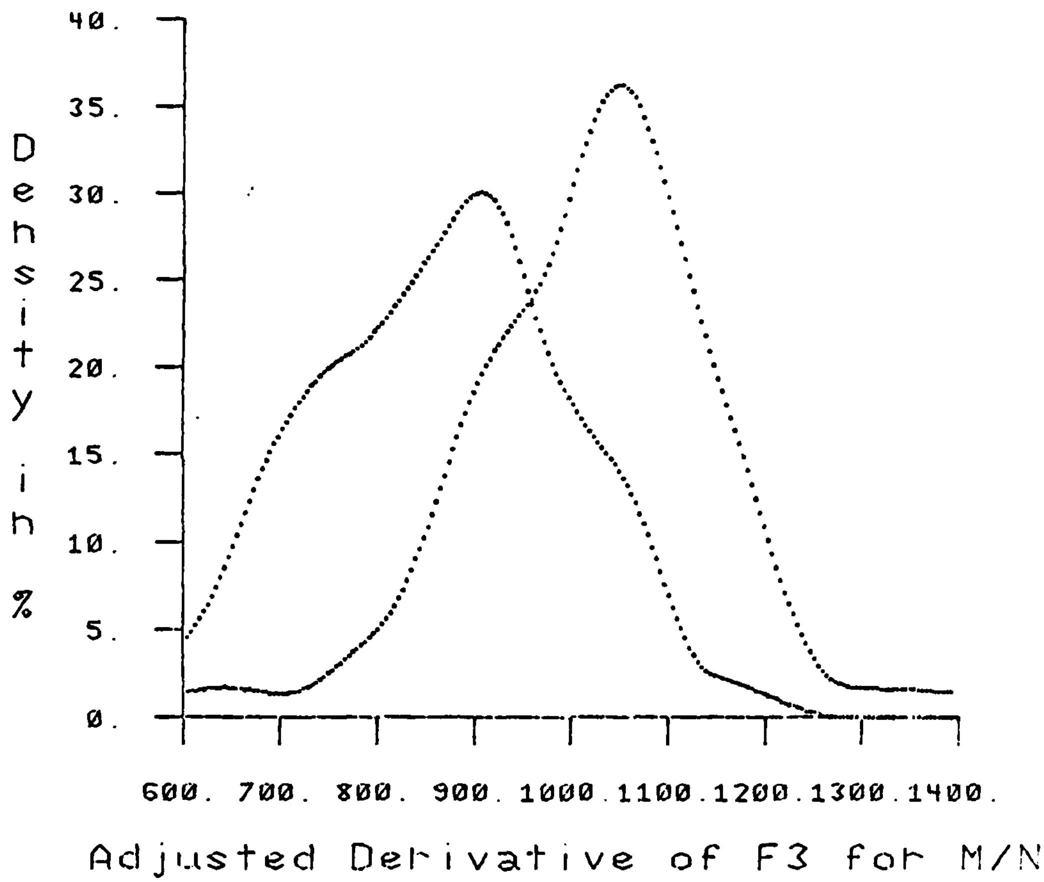


Fig. 2. DERIVATIVE IS ADJUSTED BY VOWEL FORMANT

As can be seen, the overlap between the distributions has been substantially decreased. A similar effect is shown for the derivative of F2 in Figures 3 and 4. While each of these features is not sufficient to separate the nasals with high accuracy, taken together with other useful features, the phoneme recognition can be quite high. Figure 5 shows the results of an APEF experiment using four features to distinguish among the three nasals. The cases considered included all the nasals preceded by any vowels. There were a total of 339 such nasals in the database. In addition to the two adjusted formant derivatives, the average F2 and F3 during the nasal were used as features. The resulting performance was 87% correct on the first choice, and 98% on the second choice. The confusion matrix for the first choice is shown at the bottom of the figure.

Figure 6 shows the improvement in performance when using context-dependent features, and when considering the phoneme context in the decision. The first set of results shows that the accuracy when using the unnormalized derivatives and averages is 84%. Using the adjusted derivatives increases the performance to 87%. Finally, we considered adding another feature - the F2-F3 distance in the frame before the nasal. This distance should be small for [NX] and larger for the other nasals. As shown, the performance actually decreases to 86%, due to the fact that this formant distance can also be affected considerably by the preceding vowel. Consequently, we considered the cases of nasals preceded by those vowels with close F2-F3 distance

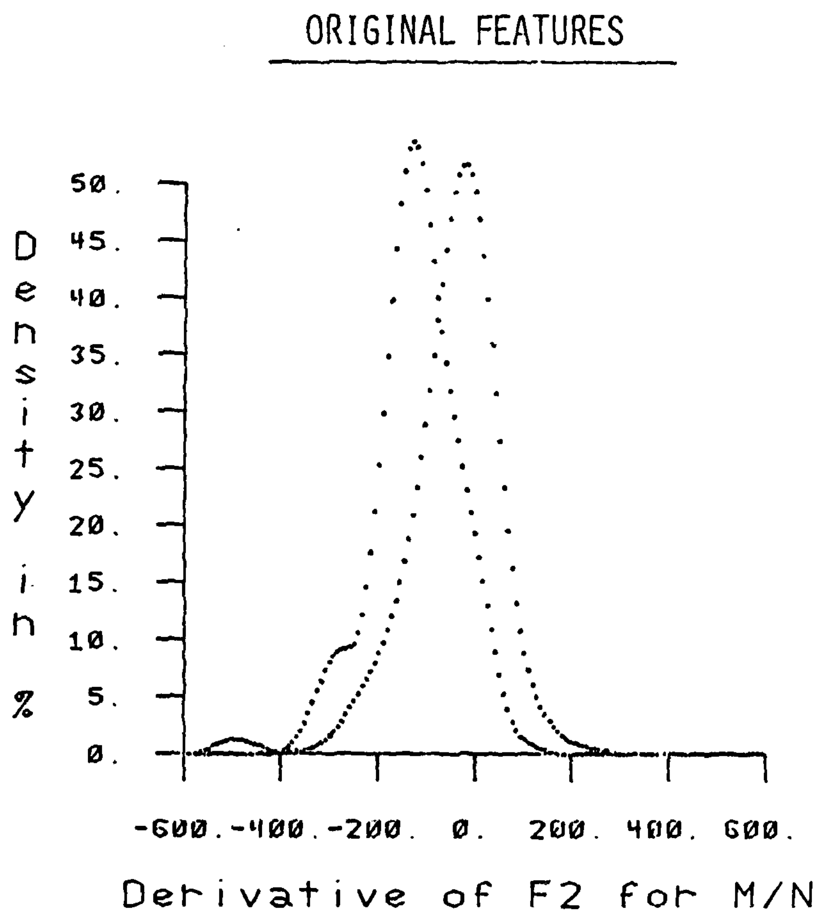


Fig. 3. RAW FEATURE HAS LARGE OVERLAP

CONTEXT-DEPENDENT FEATURES

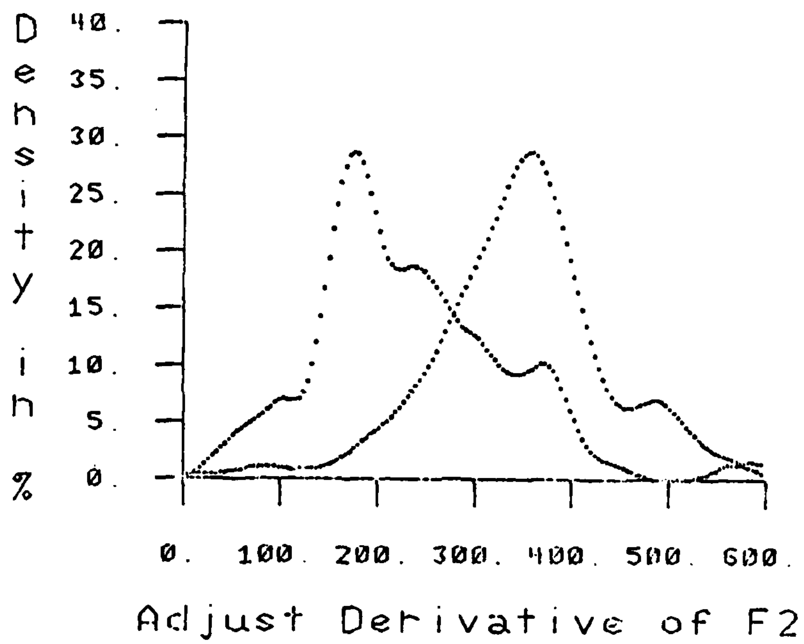


Fig. 4. DERIVATIVE ADJUSTED BY VOWEL FORMANT

USE OF CONTEXT-DEPENDENT FEATURES

CONTEXT: (VOWEL) (NASAL)

FEATURES:

1) ADJUSTED F2 DERIVATIVE
2) ADJUSTED F3 DERIVATIVE
3) AVERAGE F2 DURING NASAL
4) AVERAGE F3 DURING NASAL

RESULT: 87% CORRECT; 98% ON SECOND CHOICE

		CORRECT		
		N	M	NX
CHOSEN	N	211	11	8
	M	21	61	2
	NX	1	1	23

Fig. 5.

IMPROVEMENT ACHIEVED BY USING PARAMETER AND PHONETIC CONTEXT

CONTEXT: (VOWEL) (NASAL)

FEATURES	PERFORMANCE (%)		
DERIVATIVES OF F2, F3 AVERAGES OF F2, F3 DURING NASAL	84		
ADJUSTED DERIVATIVES OF F2, F3 AVERAGES OF F2, F3	87		
		PRECEDED BY	
	ALL	IY, EY, AY, OW, OY, AW, ER, AXR	OTHERS
ADJUSTED DERIVATIVES OF F2, F3 AVERAGES OF F2, F3 F2 - F3 DISTANCE BEFORE NASAL	86	92	90

Fig. 6.

([IY,EY,AY,OW,OY,AW,ER,AXR]) separately from all other vowels. As shown, the performance in these two cases when considered separately increased to 92% and 90% respectively. This recognition performance among the nasals is quite high, compared to results reported in the literature, and compared to results previously obtained by us.

We have shown that by performing the recognition under separate hypotheses for the context, the separation of phonemes within a class can be improved significantly. Although the APR program cannot reliably determine the phonetic context, the word matching component in the speech recognition system has available the complete spelling of any word being hypothesized by the rest of the system. It can then use only the scores assigned under the correct hypothesis for the context. This same logic also applies to a program trying to determine the phonetic identity of a sentence (e.g., a phonetic vocoder), since when evaluating any phoneme hypothesis, the program can take into account the previous phonemes in the theory under consideration.

REFERENCES

[1] R. Schwartz.
Acoustic-Phonetic Experiment Facility for the Study of Continuous Speech.
In _IEEE International Conference on Acoustics, Speech and Signal Processing_, pages 1-4. Philadelphia, PA, April, 1976.

APPENDIX 1

SENTENCE LIST AND PHONETIC ANALYSIS

FOR NEW DATABASE

Final ARPA/ONR sentences, March 19, 1982

1 Halve the height of the display window.

2 Show the result below and to the left of the graph.

3 I think someone's made a mistake.

4 Draw an equilateral triangle around the equation.

5 How far is the throughway from the railroad terminal?

6 Compute a spectrogram and display it.

7 Which utterance contains the fewest sonorants?

8 These two equations are inconsistent.

9 Reduce the abscissa to a twelfth of its present length.

10 Exchange the positions of the graph and the text.

11 Is there a good manuscript on this topic in the library?

12 I want to edit the latest version of the proposal.

13 Before showing that data, check its security classification.

14 There should be a subroutine to do that automatically.

15 Whose decision was that?

16 Find the average of the smallest values.

17 Calculate the sample's standard deviation.

18 What happened to the missing data?

19 What's the temperature in New York City?

20 Is there a good flight available in the reverse direction?

21 Is the weather forecast for tomorrow better?

22 Can the budget be stretched to cover these costs?

23 Plot both functions in the same figure.

24 Switch the order of these two paragraphs.

25 Put the title at the top of the page.

26 Capitalize only the proper names.

27 There should be a comma after the third word in this sentence.

28 Remove the opening quotes from the next three phrases.

29 How much was budgeted for travel in the second quarter?

30 When is the next progress report due in Washington?

31 Plot one function immediately above the other.

32 Isn't there some other way to show the structure in the data?

33 Indent alternate lines in this paragraph.

34 Show me all the expenses approved so far.

35 How many purchase orders are outstanding?

36 Why has his trip report not been submitted?

37 Are there any other late trip reports?

38 Can we close the books on the project's first phase now?

39 The RFP contained a detailed work statement.

40 Who do we have free to work on this task?

41 Reschedule the remainder of the tasks as well.

42 Who's project manager for this effort?

43 Did the accounting system screw up again?

44 Has a critical path analysis been done for this job?

45 Which jobs are furthest behind schedule?

46 Arrange a meeting of all task managers for Tuesday.

47 Have the replacement parts been delivered yet?

48 Top management will have to decide on that question.

49 The system crash destroyed most of our records.

50 Were the records archived, and could we retrieve them?

51 You'd better consult a lawyer about our liability.

52 Print up a financial summary sheet.

53 Can't we avoid an overrun on the rest of the tasks?

54 Print the last four messages.

55 Is there any mail in my box?

56 Show me message seventy two.

57 Delete the first twenty eight messages.

58 Send message fifty seven to everyone in the group.

59 Clear the screen.

60 Open my mail file.

61 Archive all deleted messages.

62 What messages have I had from ISI this year?

63 Print a draft of this chapter.

64 How many pages still need editing?

65 Who at BBN is working on VLSI?

66 Where and when is the ARPA review meeting?

67 Close this file.

68 A hundred and thirty one pages is too long.

69 Tell Dwayne we don't anticipate a cost overrun.

70 Who's on vacation now?

71 Who deleted the backup files?

72 There's room for five more lines in the footnote.

73 Is there a concept called "Mailing Address," or something close?

74 What concepts have roles with role-names like "Nickname?"

75 Show me the generic concept called "Employee."

76 I can't fit a new label below it.

77 Could you move it up?

78 Now make an individual employee concept.

79 The first name is "Gordon," and the last name is "Hook."

80 His Social Security number is 2 5 4, 3 6, 7 9 zero 8.

81 Is there a role on employee called "Retirement Fund?"

82 How about a role called "Pension Program," or "Pension Plan?"

83 I'd like to see the structure below "Employee Benefits."

84 Empty the screen and save the current display.

85 What's the book say on old age benefits?

86 Is it a dollar amount?

87 Set the value of old age benefits to twenty five thousand dollars.

88 Can you display just the role-set?

89 O.K., use the old role.

90 No, change that.

91 Display the role-set and the old role.

92 Now I need some figures on overhead and IR&D.

93 I'd like to see the representation of "Companies."

94 Thanks.

95 Now show me the concept for "Persons."

96 I need your help on layout decisions.

97 Change the role-name to "Residence."

98 Could you put the role-value map here.

99 Redefine this concept to include individuation.

100 Insert a telephone number of 6 4 3, 6 6 3 8.

101 That should read twelve thousand and three, not eleven thousand.

102 Make that forty nine hundred.

103 There should be seventeen entries.

104 Twenty K sounds about right.

105 Order some more Form fourteen seventy threes.

106 Type up six copies of the abstract.

107 A block is two fifty six words, right?

108 Change this entry to eight thousand.

109 Would you put this box to this side.

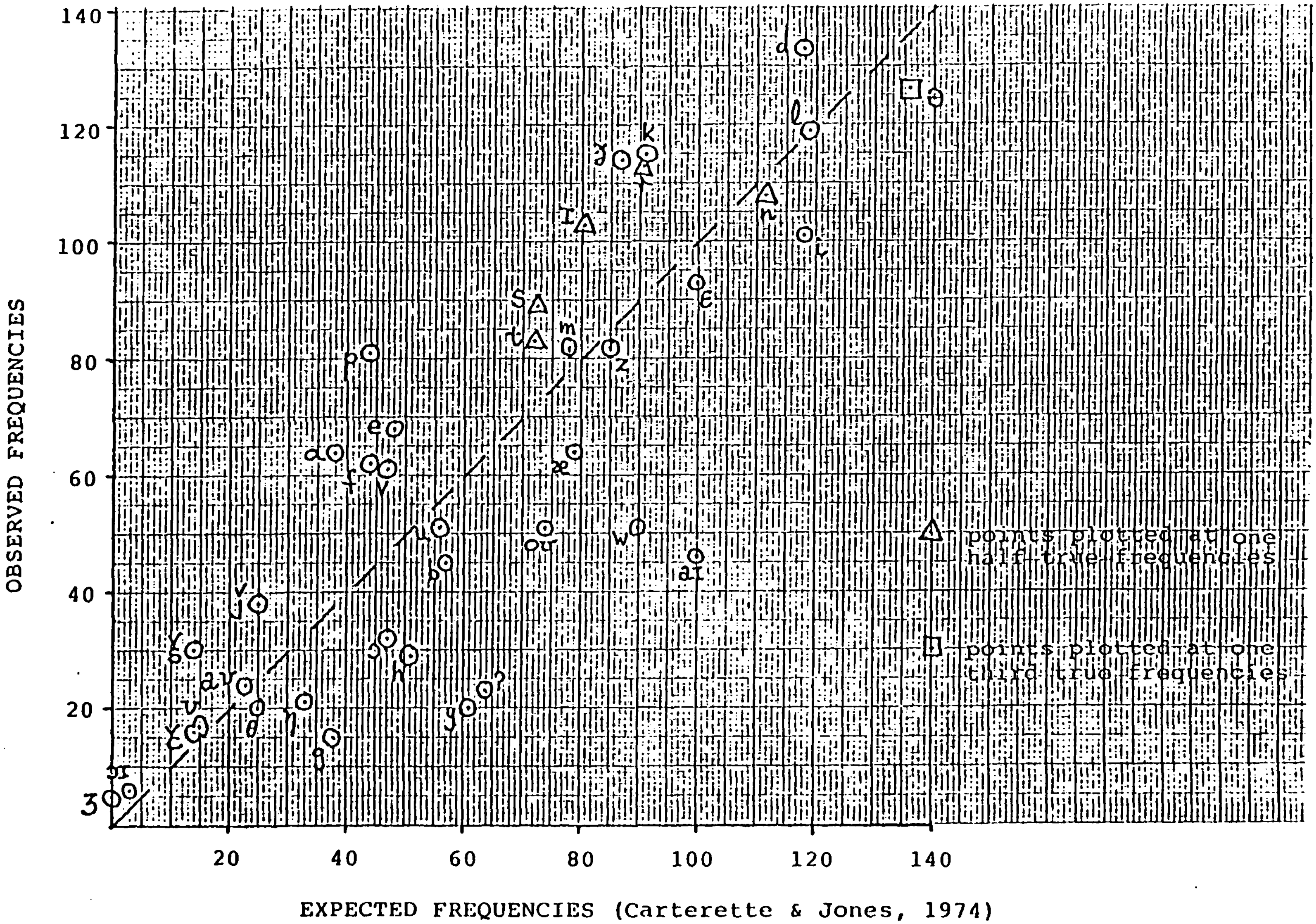
110 Interchange these two tables.

PHONEME COUNTS FOR SR-SENTS OF 19 MAR 82 (3115 TOKENS OF 49 TYPES)
(34 OF THE 3115 TOKENS APPEAR TWICE IN THE LIST => TOTAL = 3149)

R = 0.949777

		OBSERVED	EXPECTED
IY		101	118
IH\	49		
IX/	157		
"IH"		206	161
EH		93	100
AE		64	79
AA		64	38
AH\	32		
ER	18		
AX/	328		
"SCHWA"		378	408
AO		32	47
UH		17	15
UW\	35		
YU/	16		
"UW"		51	56
EY		68	48
AY		46	100
AW		24	23
OY		6	3
OW		51	74
P\	75		
URP/	6		
"P"		81	44
T\	138		
URT/	28		
"T"		166	144
K\	108		
URK/	7		
"K"		115	91
Q=?		23	64
B		45	57
D\	89		
URD	16		
DX/	28		
"D"		133	118
G		15	38
CH		16	14
JH		38	25
M		82	78
N		216	223
NX		21	33
F		62	44
TH		20	25
S		178	145
SH		30	14
V		61	47
DH		114	87
Z		82	85
ZH		5	0!
HH		29	51
L		119	119
R\	136		
ER	18		
VR/	72		
"R"		226	181
W		51	90
Y\	4		
YU/	16		
"Y"		20	61

CORRELATION COEF. R = 0.949777



APPENDIX 2

DESCRIPTION OF USER LIBRARY

BITGRAPH

CLEAR_HOLE

Fortran Subroutine
Source: DRA1:[SPEECH.LIBRARY]BGFNS.FOR
Subprogram definition:
    subroutine CLEAR_HOLE(x_min,x_max,y_min,ymax)
    Fortran subroutine to clear a rectangular window with horizontal
    x_min to x_max and vertical y_min to y_max.

Additional information available:

FULL DESCRIPTION

BITGRAPH

LENGTH_OF_INTEGER

Fortran subroutine
Source: DRA1:[SPEECH.LIBRARY]BGFNS.FOR
Subprogram definition:
    integer function LENGTH_OF_INTEGER(n)
    Fortran function to find the length of integer when printed in
    ascii characters (i.e., the number of characters). Works for
    integers from 0 to 9999.

Additional information available:

FULL DESCRIPTION

BITGRAPH

RESTORE_CURSOR

Fortran subroutine
Source: DRA1:[SPEECH.LIBRARY]BGFNS.FOR
Subprogram definition:
    subroutine RESTORE_CURSOR()
    Fortran subroutine to restore the previously saved cursor
    position.

Additional information available:

FULL DESCRIPTION

BITGRAPH

SAVE_CURSOR

Fortran subroutine
Source: DRA1:[SPEECH.LIBRARY]BGFNS.FOR
Subprogram Definition:
    subroutine SAVE_CURSOR()
    Fortran subroutine to save the current cursor position.

Additional information available:

FULL DESCRIPTION

BITGRAPH

SET_WINDOW

Fortran Subroutine
Source: DRA1:[SPEECH.LIBRARY]BGFNS.FOR
Subprogram Definition:
    subroutine SET_WINDOW(m,n)
    Fortran subroutine to set the clipping window (i.e., top and bottom margins) for a BitGraph terminal in VT100 mode. Can be used for a test window that is separate from the graphics window.

Additional information available:

FULL DESCRIPTION

COMMAND

NONE

COMPUTATION

SORTA

Fortran Subroutine
Source: DRA1:[SPEECH.LIBRARY]SORT.FOR.
Subprogram Definition:
    SUBROUTINE SORTA(ARRAY,N)
    Subroutine to sort a real array into "a"scending numeric order.

Additional information available:

FULL DESCRIPTION

COMPUTATION

SORTD

Fortran Subroutine
Source: DRA1:[SPEECH.LIBRARY]SORT.FOR.
Subprogram Definition:
    subroutine SORTD(array,n)
    Subroutine to sort a real array into "d"escending numeric order.

Additional information available:

FULL DESCRIPTION

COMPUTATION

SORTATWO

Fortran Subroutine
Source: DRA1:[SPEECH.LIBRARY]SORT.FOR.
Subprogram Definition:

subroutine SORTATWO(array1,n,array2)
Subroutine to sort a real array into "a"scending numeric order, and to sort a second array similar to the sorting of the first array.

Additional information available:

FULL DESCRIPTION

COMPUTATION

SORTDTWO

Fortran Subroutine
Source: DRA1:[SPEECH.LIBRARY]SORT.FOR.
Subprogram Definition:
subroutine SORTDTWO(array1,n,array2)
Subroutine to sort a real array into "d"escending numeric order, and to sort a second array similar to the sorting of the first array.

Additional information available:

FULL DESCRIPTION

DEBUGGING

NONE

1_0

GENERAL_INFORMATION

Waveform files (header and data) consist of sequential 16-bit integers, and should therefore always be read into and written from INTEGER*2 arrays (the default in VMS Fortran is INTEGER*4). All arguments, parameters, and variables indicated below to be 'integer' (and not specifically labelled INTEGER*2) should be assumed INTEGER*4.

Returned status codes are Fortran status codes for JOPENWAV and JCLOSEWAV (see page 7-2 of VAX-11 Fortran Users' Guide); 0 => success.
Returned status codes are RMS status codes for all other functions; .true. => success

1_0

HEADER_INFORMATION

BBN SAMPLED DATA FILE HEADER FORMAT (256 16-bit words)

```
This describes the BBN ILS-compatible Sampled Data File Header Format
The complete ILS header format is given in:
  ILS Programming Guide - Summary of ILS Subroutines (Version 3.0),
  October 1, 1980, Signal Technology, Inc., Santa Barbara, Calif.
  (document does not have unique page numbers - see approx page 16)
The header is defined to be 256 16-bit (INTEGER*2) words long,
  immediately in front of the consecutive data samples.
  (See 'IFRMAT' below for data sample format. Normal unpacked data
  sample format is 12-bit 2's complement samples, stored one per 16-bit
  (INTEGER*2) word, right justified, sign extended.)
In order for ILS routines to interact with this 256-word header,
  ILS routine 'ILS' must be modified: 'NCWFH' must be initialized to 256
  Motivations for increasing the header length from 64 to 256 words
  are as follows:
   1) 4 additional words of BBN header format can be utilized
      with no danger of future ILS use of currently unused words
      in 1-64 range.
   2) Header length of 256 allows higher sampling rate (according
      to DACS User Manual (page A2FIL-45)), since buffer boundaries
      will coincide with (256-word) block boundaries.
In the table below, the 256 words are referenced as 1-256, and are
 assumed input into an INTEGER*2 array of length 256.
```

KField 3/31/81

LOC	NAME	DESCRIPTION
1-5	--	(unused)
6	NSPBK	Number of sampled data blocks (256 16-bit words per block) in file
7-59	--	(unused)
60	IFRMAT	Flag set to 50 if samples are 8-bit, log quantization. Flag is 0 if 12-bit samples, stored one per 16-bit word, right-justified, 2's complement,

		sign extended.
61	IPWR	Power of ten multiplier for ISF
62	ISF	Sampling Frequency (see IPWR)
63	DATYPE	= -32000, indicating sampled data (= -29000 if analysis data) (= -30000 if record data)
64	SDINIT	= 32149 if sampled data file initialized
65	NSMPHI	High 16-bits of number of samples in file
66	NSMPLO	Low 16-bits of number of samples in file (Number of samples = (32768.*NSMPHI)+NSMPLO)
67	MAXCUR	Current maximum unsigned amplitude in file
68	MAXORG	Original maximum unsigned amplitude in file (For MAXCUR and MAXORG, -1 => unknown)
69-256	--	(unused)

I_O

JOPENWAV

```
integer function JOPENWAV(jfn,filnam,modestr,iperod,nsamps,ifrmat)
Open a sampled data file for input or output.
```

Additional information available:

FULL DESCRIPTION

I_O

JSINWAV

```
integer function JSINWAV(jfn,array,nsamps,istat)
Read next sequential interval of samples.
```

Additional information available:

FULL DESCRIPTION

I_O

JRINWAV

```
integer function JRINWAV(jfn,array,nsamps,sampno,istat)
Read samples at a random starting place
```

Additional information available:

FULL DESCRIPTION

I_O

JSOUTWAV

```
subroutine JSOUTWAV(jfn,array,nsamps,istat)
Write next sequential interval of samples.
```

Additional information available:

FULL DESCRIPTION

I_O

JROUTWAV

```
subroutine JROUTWAV(jfn,array,nsamps,sampno,istat)
Write samples at a random starting place.
```

Additional information available:

FULL DESCRIPTION

I_O

JRSAMPNO

```
integer function JRSAMPNO(jfn)
Read number of next sequential sample.
```

Additional information available:

FULL DESCRIPTION

I_O

JSSAMPNO

```
integer function JSSAMPNO(jfn,sampno)
Set number of next 'sequential' sample.
```

Additional information available:

FULL DESCRIPTION

I_O

JCLOSEWAV

```
integer function JCLOSEWAV(jfn)
Close a sampled data file, updating header with sample count
        and sample data block count.
```

Additional information available:

FULL DESCRIPTION

MISCELLANEOUS

BBNKBINT

```
Fortran Subroutine
Source: DRA1:[SPEECH.LIBRARY]BBNKBINT.FOR.
    This code is originally from the DACS package provided by STI.
    It has been modified for use at BBN.
Subprogram Definition:
    subroutine BBNKBINT(ichar)
    Fortran subroutine to request an asynchronous read from the
    terminal with no echo.
```

Additional information available:

FULL DESCRIPTION

MISCELLANEOUS

BBNKBDIN

```
Fortran Subroutine
Source: DRA1:[SPEECH.LIBRARY]BBNKBINT.FOR.
    This code is originally from the DACS package provided by STI.
    It has been modified for use at BBN.
Subprogram Definition:
    subroutine BBNKBDIN
    Fortran subroutine to cancel a request made by BBNKBINT.
```

Additional information available:

FULL DESCRIPTION

MISCELLANEOUS

BBNKBCHK

```
Fortran Logical*1 Function.
Source: DRA1:[SPEECH.LIBRARY]BBNKBINT.FOR.
    This code is originally from the DACS package provided by STI.
    It has been modified for use at BBN.
Subprogram Definition:
    logical*1 function BBNKBCHK()
    Fortran logical*1 function which tests if a character has
    been typed after the enabling of BBNKBINT.
```

Additional information available:

FULL DESCRIPTION

MISCELLANEOUS

BLKTRNB

```
Fortran callable Macro Subroutine.
Source:     DRA1:[SPEECH.LIBRARY]BLKTRN.MAR.
Subprogram Definition:
    call BLKTRNB(src,dst,nbytes)
    Move "nbytes" bytes from array "src" to array "dst".
```

Additional information available:

FULL DESCRIPTION

MISCELLANEOUS

BLKTRNL

See FULL DESCRIPTION of BNKTRNB.

MISCELLANEOUS

BLKTRNQ

```
    See FULL DESCRIPTION of BNKTRNB.

MISCELLANEOUS

  BLKTRNW

    See FULL DESCRIPTION of BNKTRNB.

MISCELLANEOUS

  CLRB

    Fortran callable Macro Subroutine.
    Source:      DRA1:[SPEECH.LIBRARY]BLKTRN.MAR.
    Subprogram Definition:
        call CLRB(barray,nbytes)
        Set "nbytes" in the array "barray" to zero.

    Additional information available:

    FULL DESCRIPTION

MISCELLANEOUS

  CLRF

    See FULL DESCRIPTION of CLRB.

MISCELLANEOUS

  CLRL

    See FULL DESCRIPTION of CLRB.

MISCELLANEOUS

  CLRW

    See FULL DESCRIPTION of CLRB.

MISCELLANEOUS

  DIR

    Fortran callable Macro Subroutine.
    Source:      DRA1:[SPEECH.LIBRARY]DIR.MAR.
        Subprogram Definition:
        call DIR(string)
        Fortran callable routine to emulate the DCL "DIRECTORY" command
        from within a program.  The file names are printed on the users
        terminal.

    Additional Information available:

    FULL DESCRIPTION

MISCELLANEOUS
```

DIRSEARCH

Fortran callable Macro subprogram, called as an Integer Function.
Source: DRA1:[SPEECH.LIBRARY]DIRSEARCH.MAR.
Subprogram Definition:
    isuccess = DIRSEARCH(uttname)
    Macro routine to search a directory for a given filename
    string. Called as a Fortran Integer Function.

Additional information available:

FULL DESCRIPTION

MISCELLANEOUS

FINDDIR

Fortran Subroutine
Source: DRA1:[SPEECH.LIBRARY]FINDDIR.FOR.
Subprogram Description:
    subroutine FINDDIR(dir_table_file,utterance,directory)
    Subroutine to find which "directory" an "utterance" is in, by
    by searching through the entries in the file "dir_table_file".

Additional information available:

FULL DESCRIPTION

MISCELLANEOUS

FOR_STATUS

Fortran Subroutine
Source: DRA1:[SPEECH.LIBRARY]FORSTATUS.FOR.
Subprogram Description:
    subroutine FOR_STATUS(status)
    Fortran subroutine which processes I/O errors.

Additional information available:

FULL DESCRIPTION

MISCELLANEOUS

INDERR

Fortran Subroutine
Source: DRA1:[SPEECH.LIBRARY]LPAINDERR.FOR.
Subprogram Definition:
    subroutine INDERR(ind,caller_string)
    Fortran subroutine to process error codes generated
    by calls to the LPA11-K Fortran Interface Routines.

Additional information available:

FULL DESCRIPTION

MISCELLANEOUS

IRUNTM

Fortran Integer*4 Function
Source: DRA1:[SPEECH.LIBRARY]IRUNTM.FOR.
Subprogram Description:
integer*4 function IRUNTM(dummy)
Fortran integer function to calculate the process elapsed CPU time. Uses the SYS$GETJPI system service.

Additional information available:

FULL DESCRIPTION

MISCELLANEOUS

WAIT

Fortran Subroutine
Source: DRA1:[SPEECH.LIBRARY]WAIT.FOR.
Subprogram Description:
subroutine WAIT(milli_seconds)
Causes a program to suspend itself for a specified number of milliseconds. The maximum wait time is 200000 msec., or about 3 1/3 minutes.

Additional information available:

FULL DESCRIPTION

PLOTTING

NONE

```
SIGNAL_PROCESSING

  AUTOC

    Fortran Subroutine
    Source: DRA1:[SPEECH.LIBRARY]AUTOC.FOR.
    Subprogram Definition:
        SUBROUTINE AUTOC(SIG,N,R,M)
        COMPUTES THE AUTOCORRELATION FUNCTION OF A SIGNAL, SIG
        WHERE THE AUTOCORRELATIONS, R(K) ARE DEFINED AS:
        R(K)=SUM OVER I=1 TO N-K-1 (SIG(I)*SIG(I+K)), K=0,1,....,M-1

    Additional information available:

    FULL DESCRIPTION

SIGNAL_PROCESSING

  CEPST

    Fortran Subroutine
    Source: DRA1:[SPEECH.LIBRARY]CEPST.FOR
    Subprogram Definition:
        SUBROUTINE CEPST(NFFT2,FR,FI)
        COMPUTES THE CEPSTRUM GIVEN THE LOG SPECTRUM.

    Additional information available:

    FULL DESCRIPTION

SIGNAL_PROCESSING

  CONVOLVE

    Fortran Subroutine
    Source: DRA1:[SPEECH.LIBRARY]CONVOLVE.FOR.
    Subprogram Definition:
        subroutine CONVOLVE(s,rfilter,nfilter,c,nconv,nskip)
        Fortran subroutine to compute a direct convolution between a
        signal and a filter.

    Additional information available:

    FULL DESCRIPTION

SIGNAL_PROCESSING

  DB

    Fortran Function
    Source: DRA1:[SPEECH.LIBRARY]DB.FOR.
    Subprogram Definition:
        function DB(var)
        Function to return 10.0*LOG10 of the input variable.

    Additional information available:

    FULL DESCRIPTION
```

```
SIGNAL_PROCESSING

  UNDB

    Fortran Function
    Source: DRA1:[SPEECH.LIBRARY]DB.FOR.
    Subprogram Definition:
        function UNDB(var)
        Function to return 10.0**(var/10.0) (the inverse of
        the DB function) of the input variable.

    Additional information available:

    FULL DESCRIPTION

SIGNAL_PROCESSING

  FFT8P

    Fortran Subroutine
    Source: DRA1:[SPEECH.LIBRARY]FFT8P.FOR.
    Subprogram Definition:
        SUBROUTINE FFT8P(N2POW,NS,X,Y)
        FAST FOURIER TRANSFORM (RADIX 8-4-2).

    Additional information available:

    FULL DESCRIPTION

SIGNAL_PROCESSING

  FFTAUT

    Fortran Subroutine
    Source: DRA1:[SPEECH.LIBRARY]FFTAUT.FOR.
    Subprogram Definition:
        SUBROUTINE FFTAUT(LOG2N,NS,S,R,SCR)
        COMPUTES AUTOCORRELATION USING FFT

    Additional information available:

    FULL DESCRIPTION

SIGNAL_PROCESSING

  FFTR

    Fortran Subroutine
    Source: DRA1:[SPEECH.LIBRARY]FFTR.FOR.
    Subprogram Definition:
        SUBROUTINE FFTR(N2POW,NS,S,TR,TI)
        FAST FOURIER TRANSFORM OF REAL VECTOR (RADIX 8-4-2).

    Additional information available:

    FULL DESCRIPTION

SIGNAL_PROCESSING
```

```
  FRXFMP

    Fortran Subroutine
    Source: DRA1:[SPEECH.LIBRARY]FRXFMP.FOR.
    Subprogram Definition:
        SUBROUTINE FRXFMP(N2POW,NS,X,Y)
        FAST FOURIER TRANSFORM (RADIX 8-4-2).

    Additional information available:

    FULL DESCRIPTION

SIGNAL_PROCESSING

  FTOA

    Fortran Subroutine
    Source: DRA1:[SPEECH.LIBRARY]FTOA.FOR.
    Subprogram Definition:
        SUBROUTINE FTOA(F,B,NF,FS,A,WORK,NP)
        FORMANT TO PREDICTOR COEFFICIENT CONVERSION ROUTINE

    Additional information available:

    FULL DESCRIPTION

SIGNAL_PROCESSING

  HAMMNG

    Fortran Subroutine
    Source: DRA1:[SPEECH.LIBRARY]HAMMG.FOR.
    Subprogram Definition:
        SUBROUTINE HAMMNG(S,N)
        ROUTINE THAT MULTIPLIES A SIGNAL, S BY A HAMMING WINDOW.

    Additional information available:

    FULL DESCRIPTION

SIGNAL_PROCESSING

  IFFTS

    Fortran Subroutine
    Source: DRA1:[SPEECH.LIBRARY]IFFTS.FOR.
    Subprogram Definition:
        SUBROUTINE IFFTS(IFLAG,NFFT2,FR,FI)
        COMPUTES INVERSE FFT OF A SYMMETRIC FUNCTION

    Additional information available:

    FULL DESCRIPTION

SIGNAL_PROCESSING

  INTONE

    Fortran Integer Function
```

Source: DRA1:[SPEECH.LIBRARY]INTONE.FOR.
Subprogram Definition:
    integer function INTONE(ihertz)
    Fortran integer function to convert from hertz to tones.

Additional information available:

FULL DESCRIPTION

SIGNAL_PROCESSING

IPITCH

Fortran Subroutine
Source: DRA1:[SPEECH.LIBRARY]IPITCH.FOR.
Subprogram Definition:
    SUBROUTINE IPITCH(S,N,R0,V,FS,IP1,IP2,IP0,SC1,SC2,DR0,DR01)
    PITCH EXTRACTION BY CENTER CLIPPING(SPEECH SIGNAL) FOLLOWED
    BY AUTOCORRELATION. DECISIONS ARE ALSO BASED ON R0,V AND HOW
    THE PRECEDING AND FOLLOWING FRAMES BEHAVE. THE ALGORITHM THUS
    OPERATES WITH A DELAY OF ONE FRAME.

Additional information available:

FULL DESCRIPTION

SIGNAL_PROCESSING

LPAUTO

Fortran Subroutine
Source: DRA1:[SPEECH.LIBRARY]LPAUTO.FOR.
Subprogram Definition:
    SUBROUTINE LPAUTO(SIG,N,R,A,NPOLE,VP)
    ROUTINE TO COMPUTE LP COEFFICIENTS FROM WAVEFORM
    USING THE AUTOCORRELATION METHOD

Additional information available:

FULL DESCRIPTION

SIGNAL_PROCESSING

LPCON

Fortran Subroutine
Source: DRA1:[SPEECH.LIBRARY]LPCON.FOR.
Subprogram Definition:
    SUBROUTINE LPCON(N,IFLAG,R,C,A,V,SCR)
    THIS PROGRAM COMBINES THE LPC SUBROUTINES
    COEFF,LPC,CTOA,ATOC,CTOAR,ETC.,INTO ONE SUBROUTINE; GIVEN ONE
    PARAMETER REPRESENTATION OF THE LPC FILTER, IT PROVIDES TWO
    OTHER REPRESENTATIONS ALONG WITH THE NORMALIZED ERROR.

Additional information available:

FULL DESCRIPTION

SIGNAL_PROCESSING

PITCH2

Fortran Subroutine
Source: DRA1:[SPEECH.LIBRARY]PITCH2.FOR.
Subprogram Definition:
```
SUBROUTINE PITCH2(S,N,RO,V,FS,IP1,IP2,IPO,SC1,SC2,DRO,DRO1,IB)
PITCH EXTRACTION BY CENTER CLIPPING(SPEECH SIGNAL) FOLLOWED
BY AUTOCORRELATION.  DECISIONS ARE ALSO BASED ON RO,V AND HOW
THE PRECEDING AND FOLLOWING FRAMES BEHAVE. THE ALGORITHM THUS
OPERATES WITH A DELAY OF ONE FRAME.
```

Additional information available:

FULL DESCRIPTION

SIGNAL_PROCESSING

POLES

Fortran Subroutine
Source: DRA1:[SPEECH.LIBRARY]POLES.FOR.
Subprogram Definition:
```
SUBROUTINE POLES(A,WORK,NPOLE,SPEC,NFFT2,FREQ,BW,AMP,NCR,
1 PF,PB,NP,IER)
COMPUTES LINEAR PREDICTION POLES BY FINDING THE ROOTS OF
THE POLYNOMIAL:
H(Z)=A(0)+A(1)/Z+A(2)/Z**2+ ... + A(NPOLE)/Z**NPOLE
```

Additional information available:

FULL DESCRIPTION

SIGNAL_PROCESSING

POLZRO

Fortran Subroutine
Source: DRA1:[SPEECH.LIBRARY]POLZRO.FOR.
Subprogram Definition:
```
SUBROUTINE POLZRO(A,B,NORDA,NORDB,X,Y,NSAMP)
ROUTINE TO POLE-ZERO FILTER THE SIGNAL X
```

Additional information available:

FULL DESCRIPTION

SIGNAL_PROCESSING

PPICK

Fortran Subroutine
Source: DRA1:[SPEECH.LIBRARY]PPICK.FOR.
Subprogram Definition:
```
SUBROUTINE PPICK(SPEC,NS,PFRQ,NP,DF)
PICK CENTER FREQUENCIES OF PEAKS FROM A MAGNITUDE SPECTRUM.
```

Additional information available:

FULL DESCRIPTION

SIGNAL_PROCESSING

PREAUT

Fortran Subroutine
Source: DRA1:[SPEECH.LIBRARY]PREAUT.FOR.
Subprogram Definition:

```
    SUBROUTINE PREAUT(RIN,ROUT,NIN,BFREQ,SFREQ)
    ROUTINE TO PREEMPHASIZE SIGNAL IN THE AUTOCORRELATION DOMAIN.
```

Additional information available:

FULL DESCRIPTION

SIGNAL_PROCESSING

SPEC

Fortran Subroutine
Source: DRA1:[SPEECH.LIBRARY]SPEC.FOR.
Subprogram Definition:

```
    SUBROUTINE SPEC(LOG2N,NSAMP,S,SP,WORK)
    COMPUTES POWER SPECTRUM OF A REAL SIGNAL
```

Additional information available:

FULL DESCRIPTION

SIGNAL_PROCESSING

VTL

Fortran Subroutine
Source: DRA1:[SPEECH.LIBRARY]VTL.FOR.
Subprogram Definition:

```
    SUBROUTINE VTL(F,B,NFORM,XL,FSL)
    ESTIMATES THE VOCAL TRACT LENGTH FROM A SET OF FORMANT
    FREQUENCIES AND HALF-BANDWIDTHS.  USED THE WAKITA-ZUE METHOD.
```

Additional information available:

FULL DESCRIPTION

SIGNAL_PROCESSING

ZCROSS

Fortran Subroutine
Source: DRA1:[SPEECH.LIBRARY]ZCROSS.FOR.
Subprogram Definition:

```
    SUBROUTINE ZCROSS(ISIG,N,NSO)
    THIS ROUTINE COMPUTES THE NUMBER OF ZERO-CROSSINGS OF A SIGNAL
    OVER A GIVEN INTERVAL.  FOR A ZERO-CROSSING, THE SIGNAL MUST
    CROSS ZERO (I.E., NOT JUST -1,0,-1).
```

Additional information available:

FULL DESCRIPTION

SIGNAL_PROCESSING

ZPHAS3

```
Fortran Subroutine
Source: DRA1:[SPEECH.LIBRARY]ZPHAS3.FOR.
Subprogram Definition:
     SUBROUTINE ZPHAS3(XIN,XOUT,N,CENTER)
     THREE POINT ZERO-PHASE FILTER
```

Additional information available:

FULL DESCRIPTION

STRING_PROCESSING

ADDDIRSTR

```
Fortran Subroutine
Source: DRA1:[SPEECH.LIBRARY]ADDDIRSTR.FOR.
Subprogram Definition:
     subroutine ADDDIRSTR(filename,new_dir_string)
     Subroutine to add a directory name string to a filename string.
```

Additional information available:

FULL DESCRIPTION

STRING_PROCESSING

ADDEXTSTR

```
Fortran Subroutine
Source: DRA1:[SPEECH.LIBRARY]ADDEXTSTR.FOR.
Subprogram Definition:
     subroutine ADDEXTSTR(filename,new_ext_string)
     Fortran subroutine to append an extention string onto a
     filename string.
```

Additional information available:

FULL DESCRIPTION

STRING_PROCESSING

APPENDSTR

```
Fortran Subroutine
Source: DRA1:[SPEECH.LIBRARY]APPENDSTR.FOR.
Subprogram Definition:
     subroutine APPENDSTR(base_string,append_string)
     Appends "append_string" to "base_string".
```

Additional information available:

FULL DESCRIPTION

STRING_PROCESSING

CHARCOUNT

Fortran Integer Function
Source: DRA1:[SPEECH.LIBRARY]CHARCOUNT.FOR.
Subprogram Definition:
    integer function CHARCOUNT(string)
    Fortran function to count the number of characters in a string.

Additional information available:

FULL DESCRIPTION

STRING_PROCESSING

CHGDIRSTR

Fortran Subroutine
Source: DRA1:[SPEECH.LIBRARY]CHGDIRSTR.FOR.
Subprogram Definition:
    subroutine CHGDIRSTR(filename,new_dir_string)
    Fortran subroutine to change the directory of a filename string.

Additional information available:

FULL DESCRIPTION

STRING_PROCESSING

CHGEXTSTR

Fortran Subroutine
Source: DRA1:[SPEECH.LIBRARY]CHGEXTSTR.FOR.
Subprogram Definition:
    subroutine CHGEXTSTR(filename,new_ext_string)
    Fortran subroutine which deletes the present extention of
    a filename and substitutes a new one.

Additional information available:

FULL DESCRIPTION

STRING_PROCESSING

CHGFILSTR

Fortran Subroutine
Source: DRA1:[SPEECH.LIBRARY]CHGFILSTR.FOR.
Subprogram Definition:
    subroutine CHGFILSTR(filename,new_file_string)
    Fortran subroutine to change the "name" portion of a filename
    string to a new name.

Additional information available:

FULL DESCRIPTION

STRING_PROCESSING

CONFIRM

Fortran Logical Function
Source: DRA1:[SPEECH.LIBRARY]CONFIRM.FOR.
Subprogram Definition:
    logical function CONFIRM(prompt_string)
    Fortran logical function which prompts SYS$INPUT (normally the user's terminal) and accepts a single character as input. A default prompt is supplied if the prompt string contains no characters. Returns .true. for (cr), (esc), Y(cr) or T(cr). Returns .false. for N(cr) or F(cr).

Additional information available:

FULL DESCRIPTION

STRING_PROCESSING

DELDIRSTR

Fortran Subroutine
Source: DRA1:[SPEECH.LIBRARY]DELDIRSTR.FOR.
Subprogram Definition:
    subroutine DELDIRSTR(filename)
    Fortran subroutine to delete the directory name portion of a filename string. Any legal VMS directory name string (logical names, subdirectories, none, etc.) is allowed.

Additional information available:

FULL DESCRIPTION

STRING_PROCESSING

DELEXTSTR

Fortran Subroutine
Source: DRA1:[SPEECH.LIBRARY]DELEXTSTR.FOR.
Subprogram Definition:
    subroutine DELEXTSTR(filename)
    Fortran subroutine which deletes the extention from a filename string. The delimeter "." between the file name and extention is also deleted.

Additional information available:

FULL DESCRIPTION

STRING_PROCESSING

FILENAME

Fortran Character Function
Source: DRA1:[SPEECH.LIBRARY]FILENAME.FOR.
Subprogram Definition:
    character*(*) function FILENAME(prompt_string)
    Fortran character function designed to prompt a user program for a filename string. It searches for the specified file, and

PROMPT

Fortran Subroutine
Source: DRA1:[SPEECH.LIBRARY]PROMPT.FOR.
Subprogram Definition:
    subroutine PROMPT(prompt_string)
    Fortran subroutine which types "prompt_string" on SYS$OUTPUT, (normally the user's terminal) and supresses the CR/LF following type-out. Useful for prompting a user program for data. Typically, "prompt_string" will be a literal string in quotation marks. Type-out begins on the current line.

Additional information available:

FULL DESCRIPTION

STRING_PROCESSING

PROMPTR

Fortran Subroutine
Source: DRA1:[SPEECH.LIBRARY]PROMPTR.FOR.
Subprogram Definition:
    subroutine PROMPTR(prompt_string)
    Fortran subroutine which types "prompt_string" on SYS$OUTPUT, (normally the user's terminal), followed by a CR/LF. Useful for prompting a user program for data. Typically, "prompt_string" will be a literal string in quotation marks. Type-out begins on the current line.

Additional information available:

FULL DESCRIPTION

STRING_PROCESSING

RCHAR

Fortran Character Function
Source: DRA1:[SPEECH.LIBRARY]RCHAR.FOR.
Subprogram Definition:
    character*1 function RCHAR(prompt_string)
    Fortran character function which types a prompt string to SYS$OUTPUT (normally the user's terminal) and reads a single character from SYS$INPUT (normally the user's terminal).

Additional information available:

FULL DESCRIPTION

STRING_PROCESSING

RSTRING

Fortran Character Function
Source: DRA1:[SPEECH.LIBRARY]RSTRING.FOR.
Subprogram Definition:
    character*(*) function RSTRING(prompt_string)

Fortran character function to type a prompt string to SYS$OUTPUT and then read a string from SYS$INPUT.

Additional information available:

FULL DESCRIPTION

STRING_PROCESSING

UPPERCHAR

Fortran Character Function
Source: DRA1:[SPEECH.LIBRARY]UPPERCHAR.FOR.
Subprogram Definition:
    character*1 function UPPERCHAR(input_character)
    Fortran function to convert a single character to upper-case.
    If already upper-case, character is returned unchanged.

Additional information available:

FULL DESCRIPTION

STRING_PROCESSING

UPPERSTR

Fortran Subroutine
Source: DRA1:[SPEECH.LIBRARY]UPPERSTR.FOR.
Subprogram Definition:
    subroutine UPPERSTR(string)
    Fortran subroutine to change each character in a string
    to uppercase.

Additional information available:

FULL DESCRIPTION